Solving the AFL Fixture Scheduling Problem

MAST90050 Scheduling and Optimisation

School of Mathematics & Statistics
The University of Melbourne

Daniel Olshansky, Leilah Taouk, Steven Nguyen, and William Batchelor

October 2023

Abstract

The Australian Football League (AFL) is an interesting sports scheduling problem due to the irregularity of the format. Creating a valid fixture is not at all a trivial problem due to the large number of constraints in place, many of which are unique to the league. Moreover, there are competing stakeholders, some of which want to maximise the value of games, and others which want to maintain fairness. In this project, we present a mathematical formulation of the problem of scheduling the regular AFL season under the dual objective of trying to maximise both the value and fairness of games. We demonstrate the difficulty of several metaheuristic methods in solving this problem, falling back to a mixed integer linear programming approach as our final solution.

Contents

1 Introduction

Established in 1897, The Australian Football League (AFL) is one of the most popular Australian Sports Leagues, attracting over 125 million spectators in 2022 [1]. Generating the annual AFL fixture is a difficult task, requiring a balance between many, often conflicting, interests. For example, competition fairness, revenue, spectator expectations, team expectations, and resource availability, among other factors, must be considered. An interconnected web of objectives and constraints is navigated to build an AFL fixture that balances multiple objectives and adheres to the unique characteristics of the league.

The league comprises of 18 teams. A regular season usually consists of 23 rounds, with 9 matches in each round, traditionally played on Fridays, Saturdays, and Sundays. Throughout the season, each team participates in 22 matches, 11 of which are called "home" games and played at their home stadium. What is interesting about the AFL fixture is its unusual competition format: a single round-robin tournament and 5 additional matches played by each team.

Creating an AFL fixture is seamlessly interpreted as a scheduling problem. Consider the stages of the problems to be rounds, jobs to be matches, and machines to be stadiums. Scheduling AFL matches can be analagous to an open-shop scheduling problem with jobs having unit processing times. Each job (match) is processed by a machine (played in a stadium). Matches are played in an arbitrary order determined by the scheduler. The aim is to, for every round, allocate matches or jobs to machines or stadiums and schedule them into time slots. In this project we aim to implement and combine a number of heuristic algorithms and a mixed integer programming formulation of the problem. We will compare the performance of each algorithm and benchmark our results against the historical fixtures in 2022 and 2023.

2 Literature Review

The general problem of scheduling a double round robin (DRR) tournament such that no team plays more than N games in a row at home or three games in a row away has been dubbed the *traveling tournament problem* (TTP) by Easton, Nemhauser, and Trick [2]. The TTP is $\mathcal{NP}$-complete when $N \leq 3$ [3]. The main objective is not temporal but is instead to minimise the total travel distance for every team over the tournament. As in many sports leagues, the time slots in the TTP are fixed. In [4], an integer programming formulation of the TTP was created and solved using a branch and price algorithm, where constraint programming was used to find fast, feasible solutions to the dual problem. Optimality was proven for an instance with eight teams. However, reaching optimality for more teams was proven to be difficult.

While the TPP focuses primarily on optimising the total travel distance of each team over the DRR, in reality, sports leagues are interested in several other performance measures. These can include fairness measures or business measures such as revenue from viewership. For example, a sequence of games is said to have a *break* in the last slot of consecutive home or away games. Minimising breaks maintains a pattern of alternating home and away games and is considered more fair. Ribeiro and Urrutia [5] considers a dual objective problem of minimising breaks while also maximising the revenue from games broadcast in the yearly Brazilian Soccer tournament. Again, an integer programming formulation was solved using a unique decomposition.

Additional constraints can also be put on the original problem that may interest tournament organisers. For example, one desirable constraint is that the games played in the second half of the season 'mirror' the games played in the first half, such that the games occur in the same order between the same teams but with reversed home and away venues. This variant of the problem was solved via a meta-heuristic in [6]. Their heuristic incorporates integrated local search and neighbourhoods, which can be created by swapping teams or rounds.

"A Framework for a Highly Constrained Sports Scheduling Problem", by Nurmi, K., *et al.* (2010) [7], introduces a framework for addressing highly constrained sports scheduling problems based on the requirements of professional sports leagues. The authors define the problem, present relevant terminology, and outline,

in detail, the constraints involved (36 in total), thus providing the reader with an extensive overview of the challenges in sports scheduling. To facilitate the development of solutions, the authors offer both artificial and real-world instances derived from various sports leagues and the algorithms implemented to solve them. They also highlight the importance of a few key scheduling considerations, such as managing home and away games and minimising breaks. Furthermore, the authors categorise the constraints, allowing the reader to understand the specific scheduling issue that each constraint addresses. The paper does, however, have limitations. While it presents optimal solutions for some instances, it lacks a comparative analysis of the performance of different solution methods. Further, it does not delve deeply into the specifics of the methods and the objectives used or provide a comprehensive evaluation of their effectiveness so that the reader better understands when specific methods work best.

Another paper, which deals exclusively with scheduling AFL matches, is "Fixture-scheduling for the Australian Football League using a Multi-Objective Evolutionary Algorithm", by Barone, L. *et al.* (2006) [8]. Barone *et al.* [8] implement a Multi-Objective Genetic Algorithm for the AFL scheduling problem. The irregularity of the AFL scheduling problem, from season to season, makes using a Genetic Algorithm attractive.

The representation of solutions is a crucial aspect of a Genetic Algorithm. [8] represents a schedule by an (n-1) sided Polygon with a centre point; each vertex represents a team, and each line segment, connecting two vertices or centre point, represents the game pairings for a round. Vertex labels are rotated to determine all rounds of the tournament. Four matrices represent a fixture for the AFL: the round-robin matrix (entry $(i, j) = r$ if team i versus team j in round r); the home team matrix, which indicates which team is the home team for each game pairing; the logical-to-actual team map; and the logical-to-actual round map matrix. Such a simple representation makes mutation straightforward. For example, the home team matrix and the logical-to-actual round matrix are mutated by swapping entries. Mutation of the logical-to-actual team map must preserve the rivalry round, for example, by exclusively switching teams connected by a line segment in the initial polygon since, by construction, the initial polygon contains a rivalry round.

In a Genetic Algorithm, individuals must be ranked according to their fitness. The selection of individuals who survive and multiply is based on the Pareto Rank; x is Pareto optimal if it is non-dominated with respect to the whole population [8]. In other words, at least one objective is sacrificed to improve any objective. Given that the Pareto Rank allows for the consideration of multiple objectives, it is well suited to AFL scheduling. [8] incorporates equity, travel, revenue, and venue distribution into their objective function. [8] presents a set of solutions, each balancing objectives differently, in a simple table. So, the reader quickly appreciates the trade-off between different objectives. [8] constructively presents their heuristic, such that the algorithm is reproducible.

An alternative algorithm is presented by [9]. They have built an AFL schedule using a 3-phase Model, where the output of each phase is the input of the next. First, opponents for home matches are determined. Next, each match is assigned to a round. Lastly, each match is allocated a kick-off time and venue. [9] developed a population-based local search heuristic, PEAST (Population, Ejection, Annealing, Shuffling, Tabu), to solve all 3 phases. PEAST is a Hybrid Evolutionary Algorithm advocated by [10].

During PEAST, each individual in the population conducts its own local search. [9] employs a Greedy Hill-Climbing Mutation (GHCM), in which all swaps are evaluated, and the best one is selected. Every so often, the worst schedule is replaced with the best one. So far, the algorithm has implemented a greedy local search algorithm and elitism. Cleverly, to avoid getting stuck in local optima, [9] incorporated shuffling, Tabu Search and Simulated Annealing into the GHCM. Additionally, PEAST applies shuffling operators to perturb solutions into potentially worse areas of the search space. In this way, PEAST ensures diversity in the population and at the same time, exploits promising regions of the search space. PEAST successfully balances exploration and exploitation, making the reader relatively confident that the search space is being well explored. This thoughtful approach reduces the risk of falling into local optima, enhancing the method's overall efficacy.

The PEAST algorithm assigns dynamic weights to hard constraints using an Adaptive Genetic Penalty Method (ADAGEN). The weighting of hard constraints is incrementally updated based on the search trajectory. At times preferring exploration, perhaps accepting infeasibility, and at times preferring exploitation. Exactly how weights are updated is not made clear by Kyngas [9]. As a consequence, the reader is unable to scrutinise this method. Therefore, a more detailed exploration of ADAGEN represents an important opportunity for further research. Where [9] excels, is in the way PEAST cleverly combines a multitude of traditional heuristics to develop an improved search method. This is achieved by combining the heuristics in a way that exploits their benefits and neutralises their weaknesses.

Wright[11] applies sub-cost guided simulated annealing to the scheduling of a tournament for Basketball New Zealand. Wright penalises constraint violations, and seeks to minimise penalties. Subcost guiding seeks to allow alterations that significantly improve particular penalties, by discounting increases in total penalty by the best improvement. Basketball New Zealand has similar oddities in their scheduling to the AFL.

3 Problem Description

AFL fixture scheduling can be considered an open job shop where the stages are rounds, machines are stadiums, and jobs are team pairings. Each job must be scheduled at least once. Our goal is to process jobs (team pairings) to specific stadiums (machines) and time slots (sequencing of jobs at each machine) in each round.

3.1 Parameters

- G - Set of locations
- C - Set of clubs
- S - Set of stadiums
- T =[0,1,2,3,4,5,6] corresponds to the time slots [Thursday Night, Friday Night, Saturday Afternoon, Saturday Evening, Sunday Afternoon, Sunday Evening, Sunday Evening]
- R - Set of rounds
- H_j - The home stadiums of team $j \in C$
- M_r - The importance level of round $r \in R$
- V_t - The value of time slot $t \in T$
- L_j - The ladder position of club $j \in C$
- F_j - The number of fans of club $j \in C$
- $Size_s$ - The size of stadium $s \in S$
- E - Set of rival teams where $E_{ij} = 1$ if teams i and j are rivals.

3.2 Decision Variables

A *Fixture* is represented as a 5-dimensional matrix where,

$$F_{i,j,s,t,r} = 1 \quad \text{If club } i \text{ plays club } j, \text{ in stadium } s, \text{ at time slot } t, \text{ in round } r \quad \text{and 0 otherwise}$$

3.3 Constraints

- Each team plays once every week (or every round).
- Each team plays 11 home games.
- Each team plays an equal number of home and away games.
- A team cannot play themselves.
- A team cannot play more than one home game against the same team.
- Each team must play every other team.
- A team must have at least a 5 day break between games.
- A team cannot play more than 3 games in a row outside their home location.
- A stadium can host at most one game each day.
- A team cannot play more than 4 away games in a row.
- There cannot be more than two simultaneous games.

Formally, the constraints are given as the following

$$\sum_{j\in C}\sum_{s\in S}\sum_{t\in T} F_{i,j,s,t,r} + F_{j,i,s,t,r} = 1 \quad \forall i \in C \quad r \in R \tag{1}$$

$$\sum_{j\in C}\sum_{s\in H_i}\sum_{t\in T}\sum_{r\in R} F_{i,j,s,t,r} = 11 \quad \forall i \in C \tag{2}$$

$$\sum_{s\in S}\sum_{t\in T}\sum_{r\in R} F_{i,i,s,t,r} = 0 \quad \forall i \in C \tag{3}$$

$$\sum_{s\in S}\sum_{t\in T}\sum_{r\in R} F_{i,j,s,t,r} + F_{j,i,s,t,r} \geq 1 \quad \forall i \in C \quad j \in C \tag{4}$$

$$\sum_{s\in S}\sum_{t\in T}\sum_{r\in R} F_{i,j,s,t,r} \leq 1 \quad \forall i \in C \quad j \in C \tag{5}$$

$$\sum_{j\in C}\sum_{s\in S}\sum_{t\in[4,5,6]} F_{i,j,s,t,r} + F_{j,i,s,t,r} + \sum_{j\in C}\sum_{s\in S}\sum_{t\in[0,1]} F_{j,i,s,t,r} = 1 \quad \forall i \in C \quad \forall r \in R \tag{6}$$

$$\sum_{j\in C}\sum_{s\in H}\sum_{t\in T}\sum_{r\in R}^{r+3} F_{j,i,s,t,r} + F_{i,j,s,t,r} \geq 1 \quad \forall i \in C \quad \forall r \in R\backslash[n, n-1] \tag{7}$$

$$\sum_{j\in C}\sum_{s\in H}\sum_{t\in T}\sum_{r_\in[r,r+2]} F_{j,i,s,t,r} + F_{i,j,s,t,r} \geq 1 \quad \forall i \in C \quad \forall r \in R\backslash[n, n-1] \tag{8}$$

$$\sum_{j\in C}\sum_{s\in S}\sum_{t\in T}\sum_{r\in R}^{r+3} F_{i,j,s,t,r} \geq 1 \quad \forall i \in C \quad \forall R\backslash[n, n-1, n-2] \tag{9}$$

$$\sum_{i\in C}\sum_{j\in C}\sum_{t\in[5,6,7]} F_{i,j,s,t,r} \leq 1 \quad \forall r \in R \quad \forall s \in S \tag{10}$$

$$\sum_{i\in C}\sum_{j\in C}\sum_{t\in[3,4,5]} F_{i,j,s,t,r} \leq 1 \quad \forall r \in R \quad \forall s \in S \tag{11}$$

3.4 Attractiveness Function

The goal is to schedule "attractive" matches. Formally, prioritising matches that are deemed entertaining. For example, matches between closely ranked, popular, or rival teams are financially rewarding. These objectives align with the AFL's ultimate objective of creating an entertaining and commercially successful fixture.

To achieve the aforementioned goal, we define an attractiveness function. The attractiveness of a match depends on the following,

- F_i and F_j - The fan base of each team.
- L_i and L_j - The ladder position of each team.
- $Size_s$ - The size of the stadium in which the game is played.
- E_{ij} - Whether teams i and j are rivals.
- G_i, G_j - Whether the teams are located in the same area.
- M_r - The importance level of the round.
- V_t - The value of the scheduled time slot.

The attractiveness function or "reward", is defined as

$$\text{reward}(i,j,s,t,r) = \frac{(1+E_{ij})\mathbb{I}(G_i = G_s)M_r V_t \sqrt{Size_s(F_i + \frac{F_j}{2})}}{\sqrt{(1+|L_i - L_j|)(L_i + L_j)}}. \tag{12}$$

The attractiveness function rewards rivalry games. Additionally, if encourages the allocation of popular games to busy time slots and large stadiums.

3.5 Objective

The objective consists of two metrics, one is the attractiveness of the fixture (A), and the other is the unfairness of the fixture (I).

The aim is to maximise our fixture's attractiveness and fairness. The attractiveness of the whole fixtures (A) is calculated by combining the attractiveness or "reward" of all scheduled matches.

$$A = \max \sum_{i \in T} \sum_{j \in T} \sum_{s \in S} \sum_{t \in TS} \sum_{r \in R} \left(\text{reward}(i,j,s,t,r) \cdot F(i,j,s,t,r)\right). \tag{13}$$

Meanwhile, to maintain fairness, we minimise the variance between each team's expected number of wins, thus making the fixture more fair and entertaining. We define the unfairness of the fixture as,

$$I = \sum_{r \in R} r \sum_{i \in T} \left(\left[\sum_{j \in T} \sum_{s \in S} \sum_{t \in TS} \sum_{r' \in \{1,\ldots,r\}} (F(i,j,s,t,r')P(i,j,s) + F(j,i,s,t,r')(1 - P(j,i,s)) \right] - \frac{r}{2} \right)^2. \tag{14}$$

where $P(i,j,s)$ represents the probability that team i wins against team j in stadium s. Heuristically, this represents a weighted sum of the variances between each team's number of won matches. To achieve a fair outcome, after r games are played, each team's expected number of wins should be close to $r/2$. The term within the brackets represents the squared deviation from $r/2$ up to round r. Then, we cumulatively sum the squared deviation from $r/2$ for each round, $r \in R$, to maintain the fairness of the fixture, as it is being

played, round by round. In later rounds, when more matches have been allocated, fairness is weighted more highly.
The final objective is to maximise the following:

$$\max\left[A - wI\right], \tag{15}$$

where w represents the importance of fairness in our fixture schedule.

3.6 Creating Soft Constraints

The metaheuristics we implement are not well suited to applying hard constraints. So, we simply penalise the violation of constraint to 'encourage' our algorithms to produce feasible solutions. We separate our constraints into two groups, *Soft Constraints* and *Critical Constraints*. The *Critical Constraint* group consists of constraints 1, 2, 3, and 4, while the *Soft Constraint* group consists of the rest, constraints 5, 6, 7, 8, 9, 10, and 11. The critical constraints are crucial in maintaining the applicability of the fixture, in reality. For example, requiring teams to play at least once a week, or ensuring that a team is not playing in two stadiums at the same time.

A secondary objective is established for the optimisation algorithms that cannot handle hard constraints. The secondary objective counts the number of violated *Soft* and *Critical* constraints, and multiplies each of them by a weight P_V and P_F, respectively. Of course, the violation of critical constraints is penalised more harshly than the violation of soft constraints.

4 Methodology

In this section, we will propose a number of different solution methods for solving the problem. Implementations of all algorithms are readily available in our GitHub repository [1].

4.1 Mixed Integer Linear Programming Methodology

First, a mixed-integer linear program (MILP) is implemented to determine an AFL fixture. AFL fixture scheduling can be effectively modelled using linear constraints. Therefore, it can be solved using a MILP, as seen in section 3. However, a challenge arises since the inputs to the MILP are subject to change. The dynamic nature of the problem must be considered. Another issue with the use of a MILP is that the objective is non-linear due to the function that represents the fairness of a fixture (14). Nonetheless, using a MILP, we can find a feasible AFL fixture relatively quickly, which may be used as an initial solution in metaheuristic algorithms or as a point of reference to assess the performance of other optimisation algorithms that can handle the quadratic nature of our objective. Therefore, we attempt to build a MILP program that can provide us with a good starting point for other algorithms, and a good point of comparison

The variables included in Section 3 are included in our MILP, with

$$F_{i,j,s,t,r} \in \{0,1\} \; \forall \, i,j \in C, s \in S, t \in T, r \in R,$$

corresponding to weather team i is hosting team j at stadium s at time slot t, in round r. Additionally, a new set of variables is included in our MILP, given by

$$D_{i,r} \in \mathbb{R}^{+} \; \forall \, i \in C, r \in R.$$

All of the constraints listed in Section 3.3 are included within the MILP as hard constraints. Additionally, an additional set of constraints is incorporated, given by,

$$D_{i,r} \geq \sum_{j \in C} \sum_{s \in S} \sum_{t \in TS} \sum_{r' \in \{1,\ldots,r\}} \left[F(i,j,s,t,r')P(i,j,s) + F(i,j,s,t,r')P(j,i,s)\right] - \frac{r}{2}, \; \forall i \in C, r \in R, \tag{16}$$

[1] `https://github.com/owowouwu/sports-scheduling`

and

$$D_{i,r} \geq \frac{r}{2} - \sum_{j \in C} \sum_{s \in S} \sum_{t \in T} \sum_{r' \in \{1,\ldots,r\}} \left[F(i,j,s,t,r')P(i,j,s) + F(i,j,s,t,r')P(j,i,s) \right], \ \forall i \in T, r \in R. \tag{17}$$

These constraints, coupled with our new variables, ensure that the new variables correspond to the deviation of the expected number of wins for each team from the average, for each round, for both the cases when a team is expected to have less wins that the average (Equation 17), as well as when they are expected to have more than the average number of wins (Equation 16).

Thus, the objective of our MILP also changes to incorporate the values of our new variables, becoming

$$\max \sum_{i \in C} \sum_{r \in R} \left[\sum_{j \in C} \sum_{s \in S} \sum_{t \in T} (\text{reward}(i,j,s,t,r) \cdot F(i,j,s,t,r)) - wrD_{i,r} \right]. \tag{18}$$

Hence, our MILP objective includes Equation 13 from our problem formulation, which incorporates the attractiveness feature of our fixture. However, given that our second problem objective (given in Equation 14), which is to minimise the sum of squares (or variance) of the teams' deviations from the average, is non-linear, it cannot be included in our MILP. Hence, in our MILP we seek to minimise the gaps between the AFL teams' expected number of wins - which preserves linearity.

While, clearly, this differs from the original problem objective, the solutions that we achieve using this method perform very well compared to our other methods, as we observe in our results section.

4.2 GRASP

The greedy randomised adaptive search procedure, or GRASP, is a metaheuristic algorithm which involves successively constructing solutions via iterating upon greedily constructed solutions to a problem [12]. Solutions are gradually built via randomly selecting from a number of candidates at each step of construction, ranked via some greedy criterion. The 'adaptiveness' of the algorithm comes from an extra step of adjusting the algorithm in a number of ways depending on implementation to maintain feasibility, optimality, or speed. The main advantage of this approach is that it results in fast solutions that are relatively good and maintain feasibility. Here we will outline the GRASP algorithm we have constructed to obtain solutions for our scheduling problem.

4.2.1 Building Greedy Solutions

First we will discuss our approach to building greedy solutions. We do this in a two stage approach of first constructing a list of matches between two teams to be played in each round, then iteratively for each round, assign timeslots and stadiums to each match using a greedy criteria.

Stage 1 - Constructing a Match List Constructing a simple round robin schedule under no constraints can be solved easily in linear time via the polygon method [13]. This method constructs a polygon with vertices representing teams, as shown in figure 1.

The red lines represent matches, and a configuration of such a polygon gives a set of matches to be played for a round. To generate a new round we simply can rotate the polygon clockwise. This guarantees a valid round robin schedule. However the AFL schedule also consists of a number of extra games played not within the round robin tournaments. To deal with these games, once we have scheduled the first 17 games that are required for the round robin, we simply re-sample the remaining rounds randomly from the existing schedule. This procedure builds a list of matches to play in each round, but may not respect some constraints such as the limit on consecutive away games.

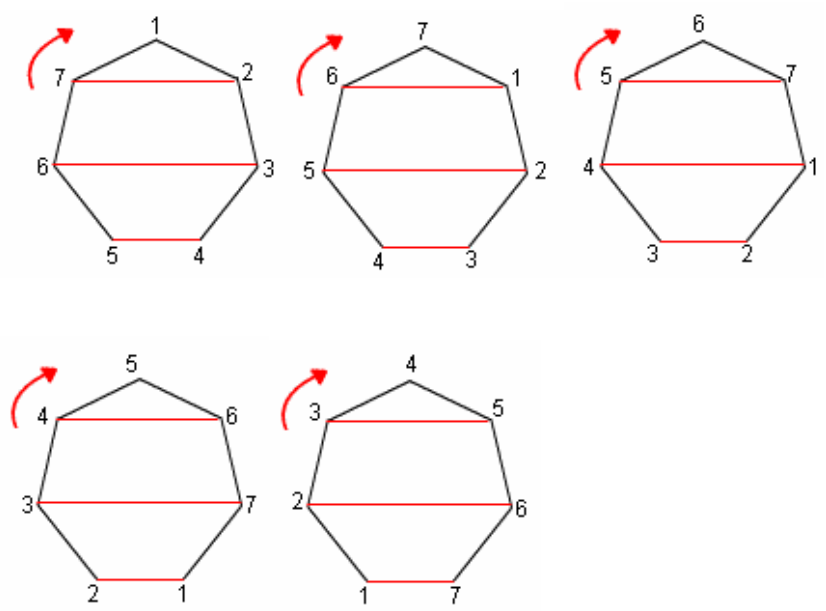

Figure 1: Polygon method for constructing round robins [14].

Stage 2 - Assigning Time-slots Once the match list has been assembled, we can now begin assigning each match a time-slot and stadium. To do this we first establish a matrix of dimension $C^2 \times R \times T \times S$ of weights with each entry initially being the attractiveness of each game. We set the weight of games which are infeasible to 0, such as those where the home team is not playing in one of their stadiums. The procedure for each round is as follows -

1. Select a random pairing of teams from the current list of matches to schedule.
2. Find the top K timeslot/stadium configurations for the team.
3. Randomly select a timeslot and stadium among the K. We discard any configuration with a weight of 0.
4. Given the whole game, update the weight matrix to maintain feasibility.
5. If we have scheduled all games for the round stop, otherwise go back to 1.

Alternatively, rather than assigning games round by round, we can assign games from our match-list by considering all rounds, time-slots and stadiums together, constructing the whole fixture in this manner rather than just round by round. Adaption is done by adjusting the weight matrix to remove infeasible games. Doing this ensures that critical constraints like a team playing twice in a round are not violated, but may also result in a situation where a valid configuration cannot be found for a specific pairing. To avoid this situation we also maintain a second weight matrix which reduces the weight of infeasible games rather than removing infeasible games altogether. In some sense, this algorithm is analogous to list scheduling, using the polygon method to construct an initial list, then shuffling games by choosing the games randomly.

4.2.2 Iteration

We implement GRASP as follows -

Algorithm 1 GRASP

Require: Max iterations $M \geq 0$

```
best_obj ← −∞
for i=1, . . . , M do
    new_solution ← ConstructGreedyFixture()
    new_solution ← AdjustSolution(new_solution)
    new_solution ← IteratedLocalSearch(new_solution)
    obj ← FixtureAttractiveness(new_solution)
    if obj < best_obj then
        best_solution ← new_solution
        best_obj ← obj
    end if
end for
```

We rely on iterated local search using neighbourhoods as defined in section 4.4. A small fix adjustment to the fixture is also made to increase the feasibility of the initial solution given to the local search algorithm, involving swapping home and away games until each team has at least 11 home games. Adaptation is done within the construction of the greedy fixtures as outlined in the previous section.

4.3 Genetic Algorithm

A Genetic Algorithm (GA) is a powerful metaheuristic that draws inspiration from evolution. Like nature's relentless drive towards improvement, a Genetic Algorithm assembles and fine-tunes a constantly evolving population with a single goal: finding the most efficient, balanced, and fair solution. In this report, this concept is applied to AFL fixture scheduling.

An individual in the population is represented as a tuple. The first entry is the fixture, the second is the objective value, and the third and fourth entries indicate the number of violated critical and soft constraints, respectively. The initial population is constructed using a combination of solutions produced by the MIP model and the greedy heuristic. Since the solution space is so large and the set of feasible solutions is comparatively small, starting with an initial set of feasible solutions helps guide the algorithm towards feasibility. More precisely, to construct an initial solution, the genesis function takes two solutions produced by the MILP model and two by the greedy heuristic. Crucial aspects of a GA are selection and reproduction. An elite selection of individuals who survive and reproduce is implemented. One parent is selected from a set of elite individuals, and the other is selected from the whole population. To determine which individuals are in the elite pool, the population is sorted according to the objective value, and the top 25% of the population is considered elite. The simple-to-implement and computationally inexpensive round crossover operator is used to produce offspring. First, a crossover round is randomly selected. Two children are produced: the first child inherits all rounds up to the cross-over round from the elite parent and the rest of the fixture from the non-elite parent; the second child inherits all rounds up to the cross-over round from the non-elite parent, and the rest of the rounds are inherited from the elite parent. Now, we have two parent fixtures and two child fixtures. Of them, the fixture with the worst objective value is removed from the population. After a specified time frame, the best fixture in the population is taken.

Algorithm 2 Round Crossover

```
function BIRTH(eliteParent, normalParent)
    crossover_point = Random integer between 0 and the number of rounds
    child1 = concatenate((eliteParent[:crossover_point], normalParent[crossover_point:]))
    child2 = concatenate((normalParent[:crossover_point], eliteParent[crossover_point:]))
end function
```

The steps in the Genetic Algorithm are presented in Figure 2 and the pseudo-code below. First, an initial population is constructed using the MIP model. For a specified period, the population evolves. Evolution

consists of the following steps: (1) Two parents are selected, with one parent being selected from a pool of elite individuals; (2) The two parents are combined through a crossover operator to produce children; and (3) The family member with the worst objective value is removed.

Algorithm 3 Genetic Algorithm

```
population ← genesis()
sorted_population ← Sorted population with respect to objective value
best_solution ← sorted_pop[0]                    ▷ Individual with the best objective value
gen_count ← 0
while Time Limit not reached do
    gen_count += 1
    parent1, parent2 ← select_parents()    ▷ Selects one individual from elite pool, and one from whole population
    Remove both parents from the population
    child1, child2 ← birth(parent1, parent2)
    family ← [parent1, parent2, child1, child2]
    sorted_family ← Sort family with respect to objective value
    sorted_family.pop()                    ▷ Remove family member with worst objective value
    Reintroduce best three family members into the population
end while
```

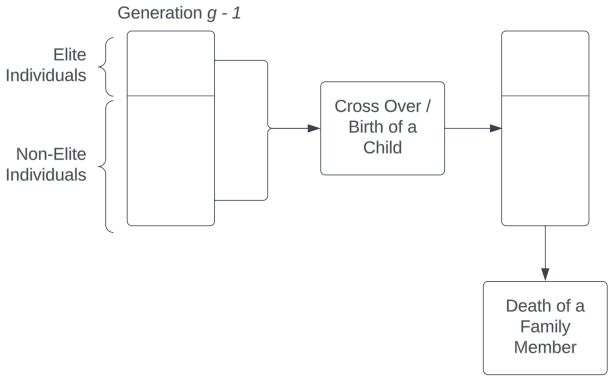

Figure 2: Flow of the Genetic Algorithm

4.4 Simulated Annealing

The next method we implemented was simulated annealing. We developed 3 neighbourhoods and used them to explore the search space. Our implementation was standard simulated annealing. The algorithm takes a solution, randomly selects a neighbouring solution from our defined neighbourhoods, and accepts it or rejects it based on the change in objective value and the current temperature. Specifically, the acceptance probability is 1 if the change in objective is positive, and $e^{\frac{\Delta}{T}}$ if the change is negative.

Temperature: We used a geometric cooling schedule. Temperature in each iteration was the temperature in the previous iteration multiplied by some constant cooling factor between 0 and 1 (though in practice, this value is closer to being between 0.9 and 1). We specified an initial temperature, a final temperature, and a number of iterations, and set the cooling factor such that the temperature would reach the desired final temperature after the correct number of iterations. Initial and final temperatures were chosen by manual tuning, and examination of the magnitude of the change in the objective value between iterations.

Before going into the neighbourhoods themselves, we present a brief discussion of the AFL season to contextualise our thinking when designing neighbourhoods. The AFL season is somewhat unusual in its arrangement.

Algorithm 4 Simulated Annealing

```
function SimulatedAnnealing(current_solution, iterations, initial_temperature, final_temperature)
    cooling_factor ← (final_temperature / initial_temperature)^(1/iterations)
    current_objective ← objective(current_solution)
    best_objective ← current_objective
    best_solution ← current_solution
    for i ← 0 to iterations − 1 do
        temperature ← initial_temperature × cooling_factor^i
        new_solution ← random_neighbour(current_solution)
        new_objective ← objective(new_solution)
        if new_objective > current_objective or
                random(0, 1) ≤ exp((new_objective − current_objective) / temperature) then
            current_solution ← new_solution
            current_objective ← new_objective
            if current_objective > best_objective then
                best_objective ← current_objective
                best_solution ← current_solution
            end if
        end if
    end for
    return best_solution, best_objective
end function
```

It's not a single round robin where each team plays every other precisely once, and it's not a double round robin where each team plays each other precisely twice. Instead every team plays one game against each other and six additional games, never playing a team more than twice. It's important to note that these additional games are spread throughout the season, and not relegated to specific rounds or the end of the season. One might think of the tournament as a single round robin with a number of bonus games, or instead as a double round robin with a number of games chosen for exclusion.

Neighbourhood 1: Home and Away swap neighbourhood. A match between teams i and j is selected, with i being the home team. We attempt to make j the home team. That is, we check the fixture and see whether any of team j's home stadiums are available in the match's round and timeslot. If so, we move the match there, and have successfully swapped to a team j home game against team i.

Neighbourhood 2: Match move neighbourhood. In this neighbourhood we select a match, and attempt to move it to a random timeslot and round, provided the stadium is available at the time. We note this move easily introduces constraint violations, especially of teams being over-booked in particular rounds. Given the difficulty we've had with building solutions that don't violate constraints, we rely on our penalties to minimize constraint violations in the eventual solution.

Neighbourhood 3: Double play swap neighbourhood. This neighbourhood seeks to explore the part of the search space introduced by the tournaments single-round robin plus bonus rounds nature. We select a pair of teams that play each other twice, and then we select another pair of teams that play each other twice. From both pairs we select a match. Then we swap the away teams for the two matches. This allows us to specifically vary who our teams are playing their bonus matches against. Swapping the away teams means we do not need to change stadiums, but will introduce constraint violations in the form of teams being overbooked in rounds.

We faced some difficulties with our neighbourhoods. Constraint violations abound. There was initially hope of developing neighbourhoods that intrinsically respected our constraints, but development of those methods stalled and we adopted a penalty approach.

To examine performance of our algorithm and see where changes needed to be made we looked at several things. The most important of these were: the objective and how it changed between iterations, how long iterations took, printouts of the fixture that was produced, and printouts of constraint violations.

We applied simulated annealing to the solutions produced by our greedy heuristic and our MILP solver, to see whether the algorithm could improve on any of those solutions.

5 Results & Numerical Analysis

We test our algorithms using the 2022 and 2023 AFL fixtures. The team wins, rankings, membership numbers, and fixtures are all available through Wikipedia's 2022 AFL season page [15], and the 2023 AFL season page [16], respectively.

5.1 MILP

We run our MILP using Gurobi's solver software, through its Python programming language API. We run the program locally, using a personal computer with 16GB of RAM, featuring 4 cores with a base speed of 1.90 GHz and 8 logical processors. We set our program's accepted optimality gap to %1. Our complete code is included in the GitHub Repository.

For the 2023 AFL fixture, our MILP finds a fixture with an attractiveness score of $-111,679$. However, given that this includes the modified equality function, we need to recalculate this using the real equality function, using which we get a worse score of $-205,122$.

However, given that there are no violated constraints, it is a feasible fixture and can be used as a starting solution for the genetic algorithm we explore.

5.2 Greedy Heuristics

Our greedy heuristic solution was found via multiple random greedy constructions based on the algorithm specified in section 4.2.1. 500 such constructions were created, with the best one resulting in a fixture attractiveness for the 2022 instance of -3,054,276, with 109 violated constraints and 1 critical constraint due to a stadium being used more than 2 times in a day. The results in table 1 show that scheduling games round by round was more effective. This is because taking the whole match list and freely scheduling pairs of teams to any round and timeslot may result in a locally optimal step causing later games to be infeasible. As a result, scheduling matches round by round was found to be better due to the fact that it was much easier to maintain feasibility within rounds, and that rounds are effectively independent of eachother for the purpose of most constraints.

While this construction is fast and contains only one critical constraint violated which can be easily fixed manually, this leaves much to be desired. Many of these violated constraints involved teams playing consecutive away games, which is to be expected due to the way we construct the schedule from a fixed match list. As such iterating upon greedy solutions may prove promising, which is why we propose to use GRASP.

GRASP was run over 5 different seeds, each with 50 iterations, and with each iterated local search step having 500 iterations. We use the 'by round' greedy heuristic to construct our initial solutions. Iterating within GRASP was shown to not improve the greedy heuristic in a meaningful manner, with the same instance resulting in an objective of -3,885,551 with 95 violated constraints and 2 critical constraints. It was found that while neighbourhood search can result in better schedules, it can be difficult to find a neighbour that is more feasible than the current schedule due to the complex dependency between games. As a result, despite the iterations upon greedy constructions, it was not found to be a substantial improvement upon a simple greedy construction.

5.3 Genetic Algorithm

The GA was executed for 20 minutes, corresponding to 23 generations. The result for the 2023 AFL Season was a fixture with an attractiveness of -3.95×10^8. However, the fixture violated over 198 critical constraints and 189 soft constraints. The initial population was constructed by the MILP model, using different parameter values to build different fixtures. Additionally, initial solutions were produced by the greedy heuristic. In doing this, the GA attempts to guide the search towards feasible solutions. The results clearly indicate that this technique, paired with the current objective function, does not effectively guide the population towards feasibility. Two changes were implemented to improve the quality of the final fixture. Firstly, when determining the individuals in the elite pool and in the selection process, rather than assessing the objective function value of each fixture solely, the population was sorted, first with respect to the number of violated constraints, then, secondarily, by the objective function. In this way, the GA is encouraged to produce a fixture that satisfies as many constraints as possible while maintaining good attractiveness. Secondly, to balance exploration with exploitation, the penalty imposed for violating constraints increased as the population evolved. In the beginning, the algorithm is more inclined to accept fixtures that violate constraints for the sake of exploration. As time passes, it becomes more important that fixtures do not violate constraints, favouring exploitation.

Now, we focus our attention on the improved GA. The GA was executed for 20 minutes, corresponding to only 17 generations. The resulting fixture had an attractiveness of -4.23×10^8 with 225 soft and 198 critical constraints violated for the 2022 instance. For the 2023 instance, a fixture with attractiveness, -3.95×10^8, with 198 critical constraints violated and 189 soft constraints violated, was generated. For the 2023 fixture, it is seen that, overall, 387 constraints are violated in the improved GA. It is highlighted that the base GA is able to execute more generations within the 20-minute time interval than the improved GA, since the improved GA must undergo the relatively computationally expensive task of checking the feasibility of every solution generated. If the base GA is executed for only 17 generations, it achieves a fixture with attractiveness 4.06×10^8 with 198 and 201 critical and soft constraints violated, respectively. Compared to this result, the improved GA outperforms the base GA for the 2023 fixture. The improved GA generates a more attractive fixture that violates less constraints. With this being said, though the solution was refined, the improved GA still produced an infeasible fixture.

It is noted that, for both instances, neither GA could improve on the solution produced by the MILP or outperform the historical AFL fixtures. An alternative reproduction function that maintains feasibility may be developed to further reduce the number of violated constraints, and perhaps determine a fixture that outperforms the MILP. Additionally, to increase diversity and escape local minima, the periodic introduction of new solutions may be incorporated, perhaps through the Greedy Heuristic presented above. Given the difficulty posed by the generation of new solutions that satisfy constraints, producing feasible immigrants proved difficult. If the Greedy Heuristic were improved such that it produced feasible solutions, it would be a promising way to generate immigrants. Another opportunity for improvement is the periodic local optimisation of a portion of the population. Here, simulated annealing could have been executed; however, to make this possible, the run time of simulated annealing must be drastically improved.

5.4 Simulated Annealing

We ran simulated annealing for 1000 iterations on ten different initial solutions. Initial solutions were produced using our GRASP and greedy algorithms. Each run of simulated annealing took approximately two hours of computation.

The results for simulated annealing listed in Table 1 were derived from fairly feasible initial solutions, with fewer than 10 violations of the critical constraints. Simulated annealing's performance in these runs was mixed. In both runs it improved on the initial objective by a small amount, and reduced the number of soft constraint violations, but increased the number of critical constraint violations.

Due to an encoding error for nine of the experiments, four for the 2022 season and five for the 2023 season, we supplied simulated annealing with very infeasible initial fixtures. Results of these runs may be found

in the appendix. For these runs, on average, the objective of the best solution discovered by simulated annealing was 2.22×10^8 more than the objective of the input solution, going from an average of -5.04×10^8 to -2.82×10^8. We saw improvement in the number of soft and critical constraints being violated, with number of soft violations going from an average of 285 to 94, and the number of critical violations going from an average of 209 to 182.

Despite seeing improvements, the result is not promising. If we assume a linear rate of decrease in the number of critical constraint violations, it would take approximately 8 hours of computation to yield a schedule violating no critical constraints. However, this assumption is unlikely to be a good one. We observed the rate of improvement decreasing, with it taking many more iterations to find a neighbour with an improved objective later in our experiments.

Simulated annealing achieved improvement upon the solutions produced via our greedy and grasp algorithms, but fell short of the results produced by a commercial solver in much less time - runs of simulated annealing took two hours, on top of the time to generate the initial solution, compared to forty minutes for Gurobi. We see two main issues in our simulated annealing approach to AFL scheduling. Most importantly, our solutions were still violating, on average, 182 critical constraints. These are the constraints that must not be violated. The fixtures aren't feasible schedules that the AFL could use. Secondly, the speed. For something as commercially important as the AFL schedule it's reasonable to throw a lot of compute at it, but taking hours to produce an infeasible schedule with a low number of iterations is not a good result. By way of comparison, Wright [11] ran a simulated annealing algorithm on a sport scheduling problem for approximately ten minutes and went through 2.5 million iterations. That's more than 10^4 times faster per iteration than our simulated annealing implementation. We aren't privy to Wright's implementation but suspect the speed difference could be attributed to three factors: language choice, implementation, and the specifics of the algorithm. Our language of choice, Python, has a reputation for being relatively slow computationally. Our implementation wasn't optimised for speed; it didn't take advantage of fast python libraries such as numpy, or utilise precomputation to avoid redundant calls of the attractiveness function. And finally, we suspect that Wright's more sophisticated simulated annealing algorithm may have been computationally advantageous, as opposed to our simple neighbourhood and penalty approach.

5.5 Comparison Between Different Algorithms

In this report, five algorithms that produce an optimal fixture have been presented and analysed. Figures 1 and 3 present the results obtained using each method. Greedy Method 1 is by far the most effective heuristic. Followed, far behind, by SA, then the improved GA. In saying this, with future work, SA and GA may be drastically improved to produce results that are comparable, and may even surpass, the results produced by the MILP and the Greedy Algorithm. This is because, theoretically, SA and GA improve existing solutions by exploring their neighbourhoods. The limiting factor in this report is the algorithm's inability to explore a feasible search space and thus produce solutions that do not violate constraints. The large search space and relatively small feasible solution space paired with a fairly complicated representation, make it very difficult to remain inside the feasible solution space while exploring the neighbourhood of a solution.

The AFL fixture, which itself obviously does not violate any constraints, was only able to be beat by the mixed integer LP for the 2023 instance. As a result none heuristic algorithms, due to the difficulty in maintaining feasibility, is able to beat the historical AFL fixture. However, we do see that our MIP is able to find a good solution that maintains good attractiveness and fairness under our criteria. This shows that with more work we may be able to more consistently produce better fixtures than the AFL.

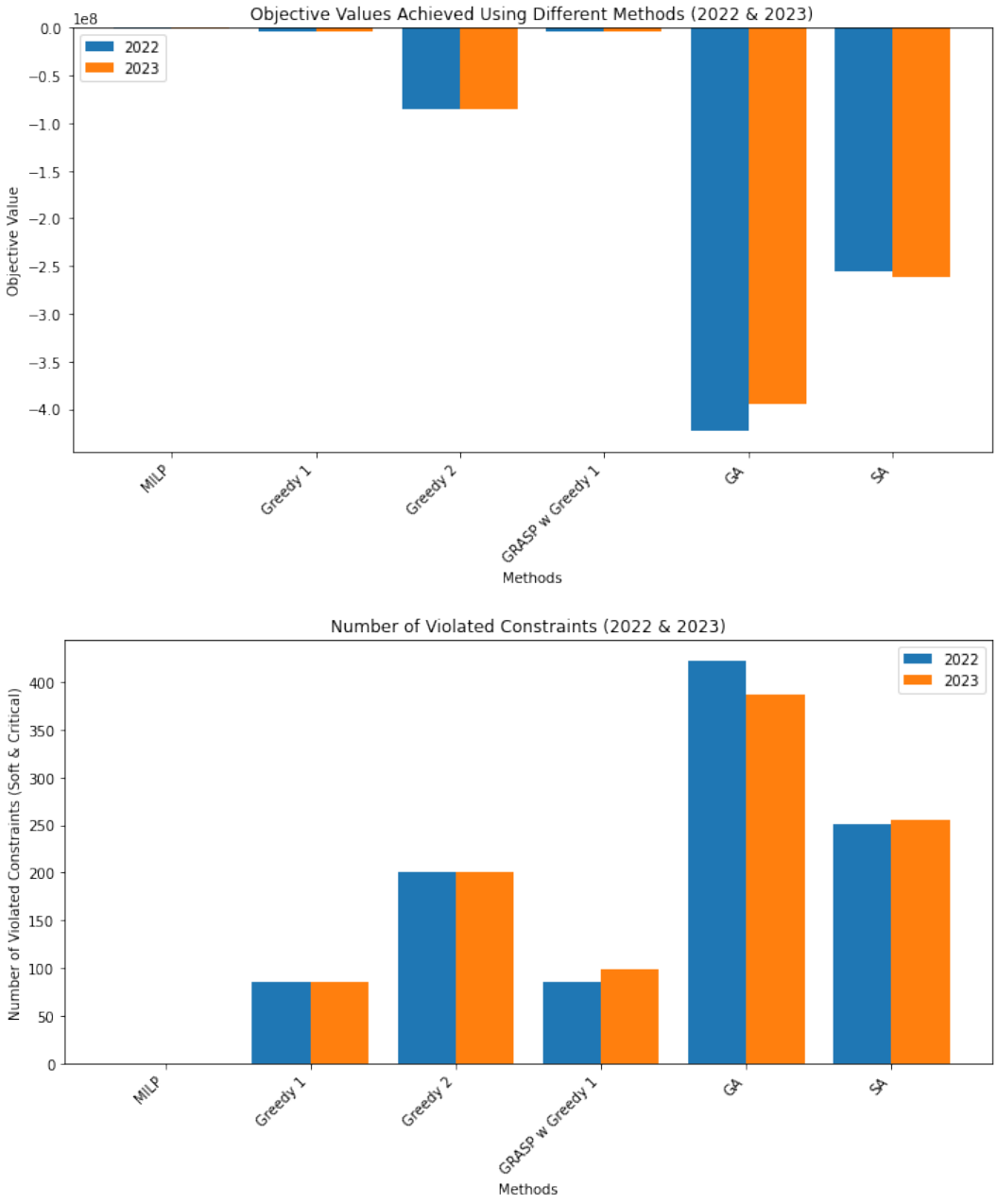

Figure 3: Visual representation of results for each method.

Model	2022 Instance			2023 Instance		
	Obj ($\times 10^3$) ↑	Soft.↓	Crit.↓	Obj($\times 10^3$)↑	Soft.↓	Crit.↓
Benchmark						
Historical AFL Fixture	-379	-	-	-378	-	-
Algorithm						
Mixed Integer LP	-448	0	0	-205	0	0
Greedy Heuristic - By Round	-3,714	84	2	-3,522	84	2
Greedy Heuristic - Whole Fixture	-85,324	117	83	-85,144	117	83
GRASP	-3,666	84	2	-3,782	97	2
Genetic Algorithm	-423,423	225	198	-394,741	189	198
Simulated Annealing - Greedy Initial Sol.	-106,587	93	11	-108,121	97	9

Table 1: Summary of results. Soft and crit denote the number of soft constraints and critical constraints violated by the solution respectively.

6 Conclusion

As demonstrated, even through the combination of multiple metaheuristic methods, scheduling a proper fixture for the AFL proves to be a difficult task. GRASP, Simulated Annealing, and the Genetic Algorithm highlighted the limits of our problem representation and ability to create neighbourhoods of the fixture that maintain feasibility. As a result, a large shortfall was trying to maintain the feasibility of a fixture. While simple fixes can be made to a fixture to improve it, such as swapping games until there are 11 home games for teach team, many other constraints are difficult to obey. Our greedy heuristics in particular highlighted that even a small number of steps can quickly lead us to the space of completely infeasible schedules, due to the complex dependency between games.

Each algorithm was also faced with its own challenges, such as runtime for simulated annealing, how to create a valid crossover for the genetic algorithm, and so on. This left the MILP approach. While it was planned to be a suitable starting point due to our objective being non-linear, it proves to be the most promising for creating any sort of valid schedule, outperforming the historical AFL fixture for one of the instances we tested on. This perhaps also highlights why such approaches are much more pervasive in the literature than simply trying out metaheuristic methods.

Moreover, while we did attempt a multi-phase approach by combining our algorithms, in the future, it may also be a good idea to break the problem down into separate stages completely, as was done in previous work. This approach was attempted with our greedy heuristic, but it would be promising to explore further. Breaking our problem down may make certain algorithms much more tractable and easier to maintain feasibility within different subproblems. More broadly, future work should focus on trying to improve local search for our problem.

In conclusion, while this project was able to implement a number of metaheuristic methods to attempt to solve the AFL fixture problem, much difficulty was encountered in making these perform well in our problem, highlighting the scale of our problem. Nevertheless show that some of these metaheuristics can result in a somewhat feasible (only a couple of critical constraints violated) fixture within a short amount of time, with clear directions for future work. As a result of our limited development of the algorithms, the black box linear programming approach performed best.

References

[1] Australian Football League. *Australian Football League 126th Annual Report*. https://www.afl.com.au/annual-reports. 2022.

[2] Kelly Easton, George Nemhauser, and Michael Trick. "The traveling tournament problem description and benchmarks". In: *Principles and Practice of Constraint Programming—CP 2001: 7th International Conference, CP 2001 Paphos, Cyprus, November 26–December 1, 2001 Proceedings 7*. Springer. 2001, pp. 580–584.

[3] Clemens Thielen and Stephan Westphal. "Complexity of the Traveling Tournament Problem". In: *Theoretical Computer Science* 412.4 (Feb. 2011), pp. 345–351. ISSN: 0304-3975. DOI: 10.1016/j.tcs.2010.10.001. (Visited on 08/29/2023).

[4] Kelly Easton, George Nemhauser, and Michael Trick. "Solving the Traveling Tournament Problem: A Combined Integer Programming and Constraint Programming Approach". In: (2002).

[5] Celso C. Ribeiro and Sebastián Urrutia. "Scheduling the Brazilian Soccer Tournament with Fairness and Broadcast Objectives". In: *Practice and Theory of Automated Timetabling VI*. Ed. by Edmund K. Burke and Hana Rudová. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2007, pp. 147–157. ISBN: 978-3-540-77345-0. DOI: 10.1007/978-3-540-77345-0_10.

[6] Celso C. Ribeiro and Sebastián Urrutia. "Heuristics for the Mirrored Traveling Tournament Problem". In: *European Journal of Operational Research* 179.3 (June 2007), pp. 775–787. ISSN: 03772217. DOI: 10.1016/j.ejor.2005.03.061. (Visited on 08/29/2023).

[7] K Nurmi et al. "A framework for a highly constrained sports scheduling problem". In: *Proceedings of the international MultiConference of Engineers and Computer Scientists*. Vol. 3. 2010, pp. 1991–1997.

[8] L. Barone et al. "Fixture-scheduling for the Australian Football League using a Multi-Objective Evolutionary Algorithm". In: *2006 IEEE International Conference on Evolutionary Computation*. 2006, pp. 954–961. DOI: 10.1109/CEC.2006.1688413.

[9] Jari Kyngäs et al. "Scheduling the Australian football league". In: *Journal of the Operational Research Society* 68 (2017), pp. 973–982.

[10] P. Preux and E.-G. Talbi. "Towards hybrid evolutionary algorithms". In: *International Transactions in Operational Research* 6.6 (1999), pp. 557–570. DOI: https://doi.org/10.1111/j.1475-3995.1999.tb00173.x. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1475-3995.1999.tb00173.x. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1475-3995.1999.tb00173.x.

[11] M.B. Wright. "Scheduling fixtures for Basketball New Zealand". en. In: *Computers & Operations Research* 33.7 (July 2006), pp. 1875–1893. ISSN: 03050548. DOI: 10.1016/j.cor.2004.09.024. URL: https://linkinghub.elsevier.com/retrieve/pii/S0305054804002461 (visited on 08/30/2023).

[12] Thomas A. Feo and Mauricio G. C. Resende. "Greedy Randomized Adaptive Search Procedures". In: *Journal of Global Optimization* 6.2 (Mar. 1, 1995), pp. 109–133. ISSN: 1573-2916. DOI: 10.1007/BF01096763. URL: https://doi.org/10.1007/BF01096763 (visited on 10/16/2023).

[13] Dalibor Froncek. "Scheduling a Tournament". In: *Mathematics and Sports*. Ed. by Joseph Gallian. Providence, Rhode Island: American Mathematical Society, 2010, pp. 203–216. ISBN: 978-0-88385-349-8 978-1-61444-200-4. DOI: 10.5948/UPO9781614442004.018. URL: https://www.ams.org/dol/043 (visited on 10/16/2023).

[14] Arunachalam Y. *Tournament Scheduling*. 2002. URL: https://nrich.maths.org/1443 (visited on 10/16/2023).

[15] Wikipedia. *2022 AFL Season*.

[16] Wikipedia. *2023 AFL Season*.

7 Appendix

Table 2: Very infeasible input simulated annealing results

Input name	Initial Objective	Initial Soft	Initial Critical	Objective	Soft	Critical
“2022-grasp1-1”	-4.77E+08	270	198	-2.64E+08	79	180
“2022-grasp1-2”	-4.86E+08	278	198	-2.77E+08	88	184
“2022-greedy1-1”	-4.85E+08	277	198	-2.88E+08	111	171
“2022-greedy1-2”	-4.73E+08	266	198	-2.56E+08	76	175
“2023-grasp1-1”	-5.02E+08	294	198	-2.84E+08	104	174
“2023-grasp1-2”	-4.86E+08	278	198	-2.82E+08	100	176
“2023-greedy1-1”	-4.85E+08	277	198	-2.79E+08	95	179
“2023-greedy1-2”	-4.73E+08	266	198	-2.61E+08	80	176
“2023-greedy2-1”	-6.68E+08	362	293	-3.43E+08	112	224